Effective Data Protection for Mobile Devices

Francisco Corella, PhD
fcorella@pomcor.com

Karen Lewison, MD
kplewison@pomcor.com

August 30, 2012

1 Introduction

Two methods are used to protect data stored in a computer device in case the device is lost or stolen. One is to store the data in tamper resistant storage, the other is to encrypt the data.

Tamper resistance is rarely used in ordinary computing devices such as smart phones, tablets, laptops or desktops, presumably because it increases the cost of a device. An exception is the use of an NFC secure element within a smart phone to store credit card data used in some payment applications. However the tamper resistance strength of such secure elements is unspecified, and, to our knowledge, none has been certified by NIST as tamper resistant (FIPS 140-2 physical level 3 or 4) [1].

Encryption is used much more frequently. Corporate laptops are often protected with full-disk encryption combined with pre-boot authentication. Since iOS 4, data stored in the iPhone and other iOS devices is encrypted under a hierarchy of keys derived in part from a PIN or a password that the user enters to unlock the phone.

The simplest way of encrypting data at rest is to use a symmetric key derived from a passcode, such as a PIN, a password, or a passphrase. But an attacker who gains physical access to the device and can extract the encrypted data from the device can mount an offline attack, trying passcodes until one is found that produces a key which successfully decrypts the data. Withstanding an offline passcode-guessing attack requires a high-entropy passcode.

Requiring the user to enter a high-entropy passcode to boot a laptop may be reasonable, but requiring a user to enter one each time he or she unlocks a smart phone is not practical. It is difficult to type a long passcode on the tiny keyboard of a smart phone, and the difficulty is compounded by the need to switch keyboards to enter different classes of characters, such as letters, digits and punctuation. In iOS, Apple addressed this difficulty by using a *hardware key* in addition to the passcode to derive the key hierarchy and protect the passcode. The hardware key is hardcoded in a hardware encryption chip and cannot be extracted by a casual attacker. However, various ways have been found of running custom code on an iOS device, and custom code can make use of the hardware key even though it cannot extract it. By making use of the hardware key it is possible to mount an offline attack against the passcode using the processor in the phone. The processor is relatively slow, but most people lock the iPhone with a 4-digit PIN, and an exhaustive brute-force attack against a 4-digit PIN takes 40 minutes on the device [2].

We propose a better way of protecting data stored in a mobile device without relying on tamper resistance. The data is encrypted under a symmetric data-encryption key, but the key is not derived from a passcode; it is a random key of sufficient length, e.g. 256 bits, stored in an online server. To retrieve the key, the user enters a PIN, which enables the device to use a key pair, which the device uses to authenticate to the server. Critically, because of how the key pair is enabled, the PIN cannot be subjected to an offline attack by an attacker who gains physical possession of the device. It is thus possible to protect the data effectively using a mere 4-digit PIN. Alternatively, the user may use a biometric such as an iris image instead of a PIN to enable the key pair, or a combination of a PIN and a biometric.

We also propose enhancements of this data protection method: the data-encryption key can be divided into pieces stored in different servers using Shamir's secret sharing scheme; and the key retrieved from the server can be hashed with the PIN, the biometric, or both, before it is used to encrypt or decrypt the data.

2 Effective Data Protection with a Simple PIN

Figure 1 illustrates the method we are proposing for protecting data stored in a mobile device with a simple PIN. The data is encrypted under a data-encryption key k. The encrypted data could be the entire persistent memory of the device; or a portion of the data stored in the device that is deemed to be particularly sensitive (such as corporate emails, or corporate documents, or credit card data used by a wallet application, or passwords saved by a browser); or just one or more encryption keys that are themselves used to encrypt portions of the data stored in the device. The key k is stored in a server that provides a data-encryption-key storage service to the device. Retrieval of the key occurs as a result of a process that is triggered by the user entering a PIN to unlock the device. The device is successfully unlocked if the process is successful and the key k is retrieved. No other means of testing the PIN (such as storing a hash of the PIN) must be available, because such other means could enable an offline attack against the PIN by an attacker who gains physical possession of the device.

The key is retrieved through a secure connection that provides confidentiality protection and server authentication. This could be a TLS connection, or a connection that encrypts data using a preshared symmetric key. The connection could be made within the carrier network to which the mobile device belongs, or across the Internet. Henceforth, the term *secure connection* will refer to a connection between the device and the server that provides confidentiality protection and server authentication.

To retrieve the data-encryption key, the device authenticates to the server using an RSA key pair [3, §8.2], whose use is enabled by the PIN entered by the user. However the PIN is not used to encrypt the private key, nor the entire pair, which would make it vulnerable to an offline brute-force guessing attack by an attacker who gains physical access to the device. Rather, it is used to *regenerate* the key pair. Thus all PINs produce well-formed key pairs, and the only way to test a PIN is to use the key pair that it produces to retrieve

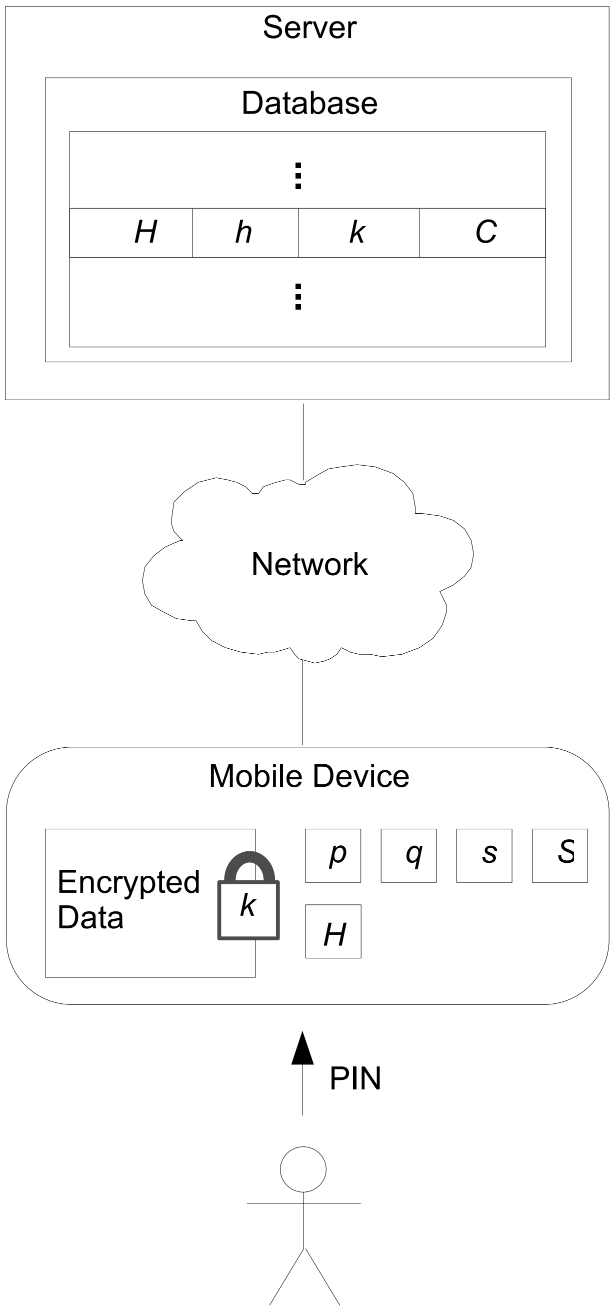

Figure 1. Effective Data Protection with a Simple PIN

the data-encryption key from the online server. The server is thus able to limit the number of guesses made against the PIN during a brute-force guessing attack to a very low number, e.g. 10, whereas the number of guesses is unlimited in an offline attack.

The data-encryption key and the RSA key pair are generated when the user sets a PIN for unlocking the device as described below in Section 2.1. A different RSA key pair may be generated later if the user changes the PIN as described below in Section 2.2.

Generating an RSA key pair involves choosing the prime factors p and q of the RSA modulus $n = pq$, and encryption and decryption exponents e and d such that $ed \equiv 1 \pmod{\phi}$, where $\phi = (p-1)(q-1)$. The private key is d and the public key is (n, e). (Therefore e is also called the public exponent and d the private, or secret, exponent.) Usually a small encryption exponent such as $e = 3$ or $e = 65537$ is chosen first, and the decryption exponent is computed as the unique integer d, $1 < d < \phi$, such that $ed \equiv 1 \pmod{\phi}$, which exists as long as e and ϕ are relatively prime. Instead, we derive the decryption exponent d from the PIN and compute the encryption exponent (using the extended Euclidean algorithm [3, Algorithm 2.107]) as the unique integer e, $1 < e < \phi$, such that $ed \equiv 1 \pmod{\phi}$, which exists as long as d and ϕ are relatively prime. The decryption exponent d must not be small, because there exists a polynomial time algorithm for computing d from the public key (n, e) if the length of d is up to approximately one-quarter of the length of the modulus [4].

The decryption exponent d is derived from the PIN, from a random seed s of sufficient length (e.g. 256 bits), from ϕ, and from the set S of *small prime factors* of ϕ, a prime factor being deemed to be small if it is less than a threshold such as 100. The process of deriving d uses a process variable $\textsc{d}$, as follows:

1. A randomized extended hash of the PIN with random seed s, of same byte length as the modulus, is computed and assigned to $\textsc{d}$. (A randomized extended hash of any byte length can be computed using the `P_hash` mechanism of the TLS protocol [5, §5], which uses a hash function such as SHA-256 [6], the HMAC mechanism [7], a secret, which in this case is the PIN, and a random seed, in this case s, to produce a cryptographic hash of the secret and the seed of the desired length.)

2. $\textsc{d} \bmod \phi$ is assigned to $\textsc{d}$.

3. If $\textsc{d}$ if divisible by one or more elements of S, $\textsc{d}$ is repeatedly divided by such elements, the result of each division being assigned to $\textsc{d}$, until no such elements remain.

The decryption exponent d is the value of $\textsc{d}$ at the end of the process. If the resulting d is not relatively prime with ϕ, or if the bit length of d is not at least $\frac{3}{4}$ of the bit length of the modulus, we start over, choosing different prime factors p, q for the modulus.

The key pair is regenerated before each use, when the user enters the PIN to retrieve the data-encryption key k and thereby unlock the device. To make it possible to regenerate the key pair, the prime factors p and q and the random seed s are retained in the device when the key pair is generated for the first time, as shown in Figure 1.[1] The set S of small

[1]Notice that p and q can be computed from the key pair [3, §8.2.2(i)], so storing p and q is no less secure than storing the key pair, or the private key and a certificate containing the public key.

prime factors of ϕ is also retained, to facilitate the process. The decryption exponent is then calculated from the PIN, s, $\phi = (p-1)(q-1)$, and S by the same process used when the key pair is generated for the first time; the encryption exponent is calculated by the extended Euclidean algorithm [3, Algorithm 2.107] as the unique integer e, $1 < e < \phi$, such that $ed \equiv 1 \pmod{\phi}$; and the modulus as $n = pq$. The private key d and the public key (n, e) are then used by the device to authenticate to the server as described below.

As shown in Figure 1, the server has a database where it stores records for the mobile devices that have entrusted their data-encryption keys to the server. The record for the device shown in the figure comprises a handle H that uniquely identifies the record[2], a hash h of the public key (n, e) computed with a cryptographic hash function such as SHA-256, the data encryption key k, and a counter C of consecutive failed authentication attempts. The handle H is generated by the server when the record is created and provided to the device, which stores it.

To authenticate, the device sends H, n, e and a nonce to the server. The server locates the record identified by H, verifies that the hash h in the record is the hash of the public key (n, e), and sends a nonce to the device. Then the device and the server separately compute m as a cryptographic hash of the concatenation of the two nonces. The device demonstrates knowledge of the private key d by computing $m' = m^d \bmod n$ and sending it to the server. The server checks if $m \equiv {m'}^e \pmod{n}$. If so, the authentication has succeeded and the server sends k to the device. The device discards d, n and e. The server discards n and e.

When the device receives k it stores it in volatile memory and uses it as needed to decrypt data while the device remains unlocked.

An attacker who has physical possession of the device can read p, q, s and H from the persistent memory of the device, and may try to guess the PIN. However, because the PIN is only used to regenerate the key pair, the attacker can only test each guessed PIN by trying to use the key pair that it produces to authenticate to the server. In other words, the PIN can be subjected to an online guessing attack, but not to an offline guessing attack. The online guessing attack can be thwarted by the server by limiting the number of authentication failures. This can be done in several ways. One way is to keep the counter C of consecutive authentication failures in the device record shown in Figure 1. When the counter reaches a limit such as 10, the server deletes the record for the device, or at least the data-encryption key k stored in the record. Since k is not stored elsewhere when the device is locked, this is equivalent to wiping out the encrypted data. The counter C is reset by a successful authentication. A second counter of total, not necessarily consecutive authentication failures can also be used as explained in US patent 8,046,827.

2.1 Setting a PIN for the First Time

The process of setting a PIN for the first time is as follows.

The device generates prime factors p and q for an RSA modulus n, computes $n = pq$, computes $\phi = (p-1)(q-1)$, and obtains the set S of small prime factors of ϕ. The device

[2] Such a handle would be called a *primary key* in database literature, but we avoid the term to prevent confusion between database keys and cryptographic keys.

generates a random seed s and computes the decryption exponent d from the chosen PIN, s, ϕ and S by the process described above. If the bit length of d is not at least $\frac{3}{4}$ of the length of the n, the device starts over, choosing different prime factors p, q for the modulus. The device computes the encryption exponent e by the extended Euclidean algorithm [3, Algorithm 2.107] as the unique integer e, $1 < e < \phi$, such that $ed \equiv 1 \pmod{\phi}$; if the computation fails because d and ϕ are not relatively prime, it starts over with different prime factors p and q. The device generates a data-encryption key k and uses it to encrypt the data. The device establishes a secure connection to the server. It sends the data encryption key k and the public key (n, e) over the connection, and demonstrates knowledge of the associated private key as explained above. (The device may also provide evidence that it is entitled to receive data-encryption-key storage service from the server.) The server computes the hash h of the public key and creates a device record containing h, k, a handle H, and a counter C initialized to 0. The server sends H to the device over the connection, and the device stores it.

Notice that encrypting the data after k has been generated can take a long time if the entire persistent memory of the device or large portions thereof have to be encrypted. This problem can be avoided by encrypting the data under one or more keys that are kept in the clear until the user chooses to set a PIN, then encrypting only those keys under k when the user sets a PIN for the first time.

2.2 Changing the PIN

The process for changing the PIN is as follows. The device generates a new key pair from the new PIN, as when setting the PIN for the first time. The device establishes a secure connection to the server, sends H, sends the old public key, demonstrates knowledge of the old private key as explained above, sends the new public key, and demonstrates knowledge of the new public key. The server uses H to locate the device record, computes the hash of the old public key received from the device, and verifies that the computed hash coincides with the hash stored in the device record. If so, it computes the hash of the new public key received from the device and substitutes it for the hash of the old the public key stored in the record.

3 Effective Data Protection with a Biometric

It is possible to use a biometric instead of a PIN to authenticate the user to the device and regenerate the key pair, thereby unlocking the device. It would be dangerous to store a biometric template in the device, where it would be exposed to an attacker who gains posession of the device. It would also be dangerous to store biometric templates in the server database, where a large number of them could be captured as a result of a security breach. But several methods of biometric authentication of a user to a device have been proposed that do not require the storage of a biometric template anywhere [8, 9, 10]. No biometric template is used in such methods. As illustrated in Figure 2, a cryptographic key, sometimes called a *biometric key*, is consistently produced whenever a genuine biometric sample is presented for authentication, using an auxiliary string. The auxiliary string is

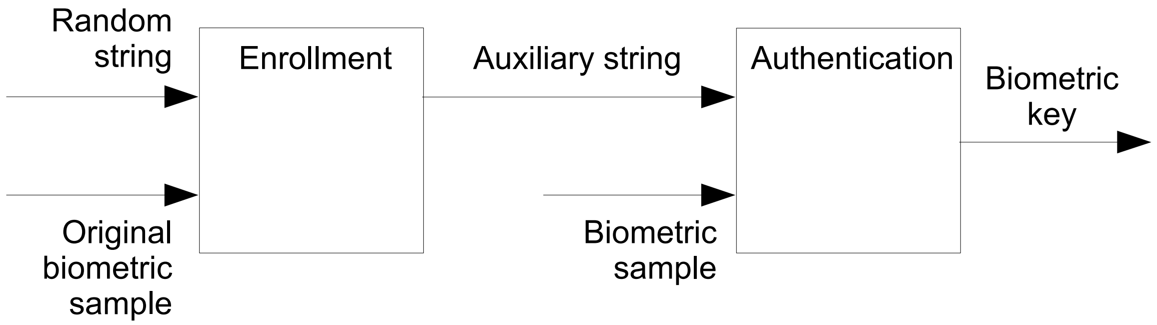

Figure 2. Generation of a Biometric key

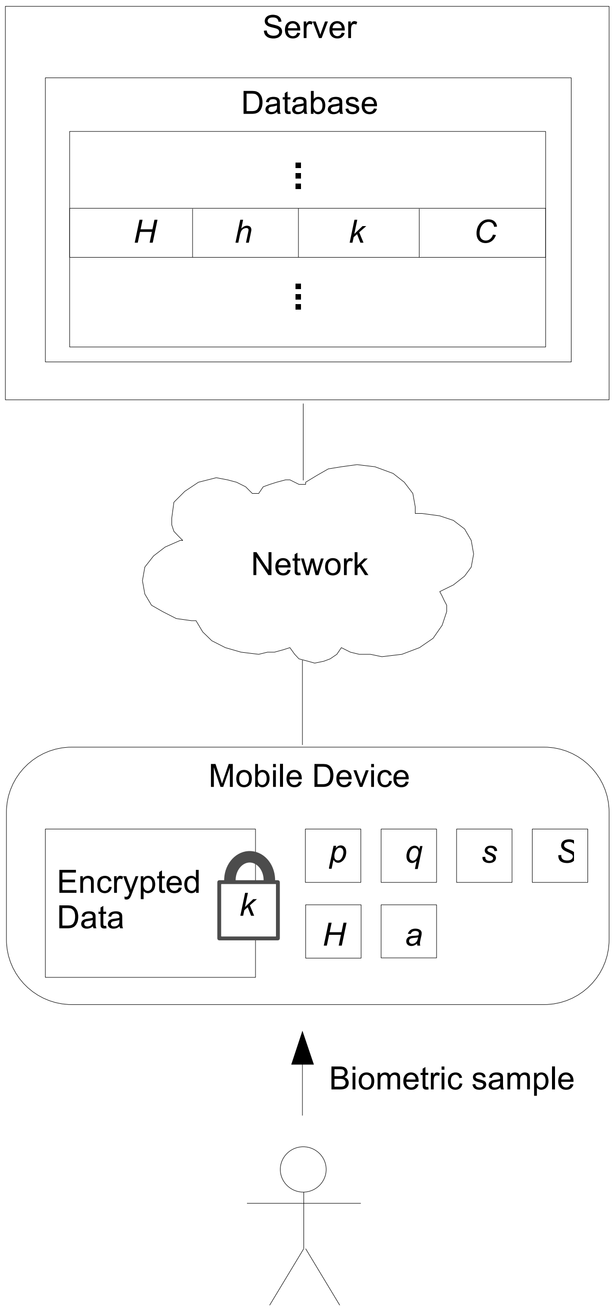

Figure 3. Effective Data Protection using a biometric

computed from an original biometric sample presented for enrollment and a random string. The original biometric sample cannot be recovered from the auxiliary string. In particular, Hao, Anderson and Daugman [10] have obtained good results using iris images.

To enable the key pair using a biometric sample instead of a PIN, the auxiliary string a is stored in the device as shown in Figure 3. When the user enters the biometric sample for authentication, the sample and the auxiliary string a are used to compute the biometric key. The key pair is regenerated as described above, except that the biometric key is used instead of the PIN.

4 Combining a Biometric and a PIN

To combine a biometric and a PIN, the PIN can be used to encrypt the auxiliary string stored in the device. A simple way of doing this is to compute a randomized extended hash of the PIN with a seed s', of same bit length as the auxiliary string, then x-or the randomized extended hash with the auxiliary string.

Another way of combining a biometric and a PIN is to use the PIN to make modifications to the function that computes the auxiliary string and to the function that computes the biometric key. Hao, Anderson and Daugman [10] suggest permuting the rows and columns of a Hadamard matrix used in both computations.

5 Enhancements

When a PIN is used, k may be hashed together with the PIN before it is used to encrypt or decrypt the data. This prevents an attacker who obtains k, e.g. by breaking into the server database, and also gains physical access to the device, from using k directly to decrypt the data. (However, the attacker may mount an offline guessing attack against the PIN, testing each candidate PIN by hashing it with k and attempting to decrypt the data.)

Similarly, if a biometric is used, the biometric key can be hashed with k. If both a biometric and a PIN are used, one or the other or both can be hashed with k.

Shamir's secret sharing technique [11] can be used to divide k into N pieces that are kept by N different servers, in such a way that the device can reconstruct k from any K of those N pieces, but any set of $K-1$ pieces reveals no information about k. This increases security, because even if $K-1$ servers collude they cannot reconstruct k. It may also increase reliability, because the device can reconstruct k even if $N-K$ servers are down. And it may increase performance, because the device can request all N pieces and use the first K that arrive.

References

[1] NIST. FIPS 140-1 and FIPS 140-2 Vendor List.
http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401vend.htm.

[2] Vladimir Katalov. ElcomSoft Breaks iPhone Encryption, Offers Forensic Access to File System Dumps, May 23, 2011. http://blog.crackpassword.com/2011/05/elcomsoft-breaks-iphone-encryption-offers-forensic-access-to-file-system-dumps/.

[3] Alfred J. Menezes and Paul C. Van Oorschot and Scott A. Vanstone and R. L. Rivest. Handbook of Applied Cryptography, 1997. http://cacr.uwaterloo.ca/hac/.

[4] Michael J. Wiener. Cryptanalysis of short rsa secret exponents. *IEEE Transactions on Information Theory*, 36:553–558, 1990.

[5] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2, August 2008. http://tools.ietf.org/html/rfc5246.

[6] NIST. FIPS PUB 180-4 Secure Hash Standard, March 2012. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[7] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication, February 1997. http://tools.ietf.org/html/rfc2104.

[8] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM Journal on Computing*, 3(1):97–139, 2008.

[9] Xavier Boyen. Reusable Cryptographic Fuzzy Extractors. In *ACM CCS 2004, ACM*, pages 82–91. ACM Press, 2004.

[10] F. Hao, R. Anderson, and J. Daugman. Combining Cryptography with Biometrics Effectively. *IEEE Trans. Comput.*, 55(9):1081–1088, 2006.

[11] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.